UNIVERSIDAD NACIONAL DE AVELLANEDA

Trabajo de Laboratorio N°1

ARQUITECTURA DE SISTEMAS DE ELABORACIÓN DE DATOS II

Alumnos:

- Bacich, Sofía
- López, Agustín Iván
- Morán, Iván

Profesor:

- Loiseau, Matías

Entrega: 24/09/2022

Segundo Cuatrimestre 2022

Índice

Objetivo

El objetivo del presente trabajo de laboratorio es diseñar un Sistema Embebido para controlar el sentido de giro de dos motores DC según la información proveniente de un sensor de proximidad.

En nuestro caso, sin embargo, se nos asignaron las funcionalidades adicionales de controlar cuatro motores DC en vez de dos y de controlar un servomotor que permitirá que nuestro sensor ultrasónico pueda “mirar” hacia la derecha y hacia la izquierda, permitiendo así una mayor autonomía de movimiento para el sistema.

El sistema deberá tener las siguientes características:

- Si la distancia recibida por el sensor es mayor a 15 cm, los dos motores deben avanzar hacia adelante.
- Si la distancia recibida por el sensor es menor o igual a 15 cm, uno de los motores debe girar hacia atrás y el otro hacia adelante.

Introducción

Como ya ha sido enumerado en los objetivos, el sistema embebido que se plantea hacer en este trabajo de laboratorio es muy simple: un auto inteligente. El mismo estará controlado por una plataforma EDU-CIAA-NXP, la cual permitirá que el sistema pueda desplazarse de manera autónoma esquivando obstáculos mediante un análisis de su entorno.

Esto será posible gracias a un sensor ultrasónico HC-SR04 que funciona emitiendo un impulso de onda ultrasónica, la cual se verá reflejada y le permitirá medir la distancia que debe avanzar. A su vez, se utilizará un servomotor para permitir que el sensor pueda rotar hacia la derecha y hacia la izquierda, y así poder sensar en esas direcciones si existe una ruta posible que permita a nuestro auto desplazarse hacia allí.

Finalmente, se empleará un par de puentes H dobles del módulo L298N para controlar la velocidad y el sentido de giro de cuatro motores DC, los cuales tendrán el objetivo de hacer que el auto pueda avanzar, retroceder y girar hacia ambos lados. El principio de funcionamiento es que tenemos cada uno de los motores “conectados” a las ruedas de cada lado del auto, las cuales vienen incluidas con los motores.

Marco Teórico

A continuación vamos a detallar los conceptos teóricos detrás de los componentes que utilizaremos en la etapa de desarrollo del trabajo de laboratorio.

EDU-CIAA

El *Proyecto CIAA (Computadora Industrial Abierta Argentina)* es un desarrollo nacional, en conjunto entre la ACSE *(Asociación Civil para la Investigación, Promoción y Desarrollo de Sistemas Embebidos)* y la CADIEEL *(Cámara Argentina de Industrias Electrónicas, Electromecánicas y Luminotécnicas)* con inicio en el 2013.

Consiste en el desarrollo de diferentes modelos de plaquetas electrónicas nacionales, industriales (es decir, que están diseñadas para soportar situaciones adversas, como la exposición a altos rangos de temperatura, protección ante picos de tensión y cortocircuitos, interferencias electromagnéticas, etc.) y de código abierto; con visión en la industria nacional y sus necesidades, tras la observación y estudio de estas, producto de varias reuniones, charlas y acuerdos con este sector.

> "El Proyecto CIAA nació en Argentina a fines de 2013 y desde entonces cientos de personas, instituciones, empresas y el estado han colaborado para que crezca. El Proyecto CIAA surgió al analizar la situación de la industria argentina y pensar cómo desde la academia se podría ayudar a su desarrollo. " [1]

Dentro de los modelos de plaquetas, se encuentran *(entre otros)*:

- CIAA-NXP
- EDU-CIAA-NXP
- picoCIAA
- CIAA-Safety
- CIAA-ACC

De los cuales nosotros utilizamos **EDU-CIAA-NXP**, concebida para la educación, como su nombre lo indica, una placa introductoria para luego utilizar la CIAA-NXP. En la Figura 1, por ejemplo, se puede ver una de estas plaquetas.

[1] *¿Qué es Proyecto CIAA?*. Proyecto CIAA. Recuperado el 19/9/2022 de: http://www.proyecto-ciaa.com.ar/index_quees.html

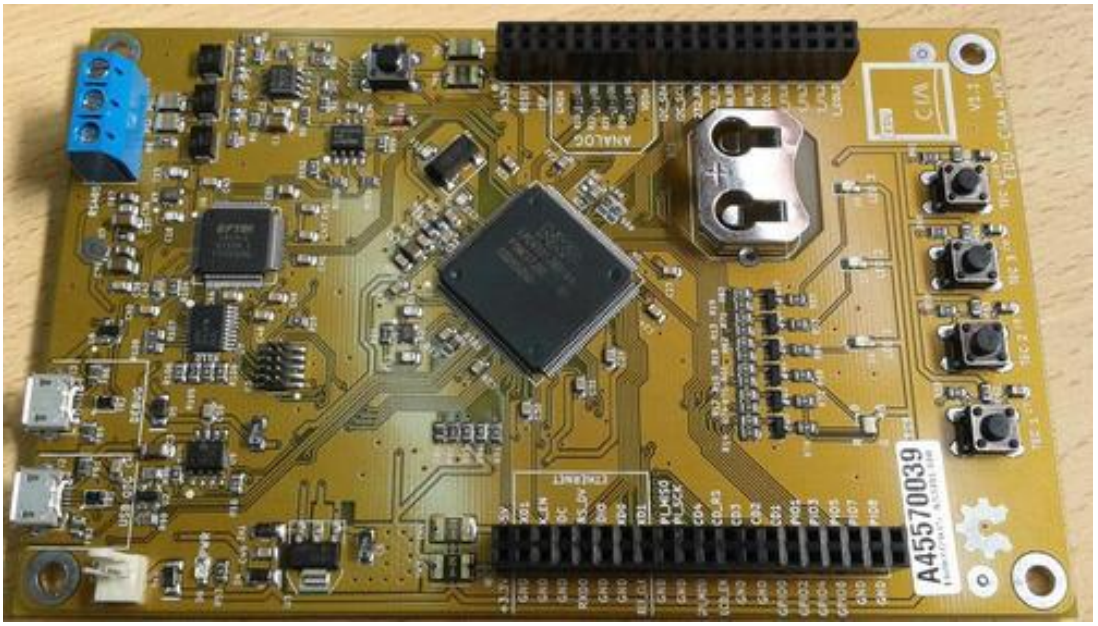

Figura 1[2]

La misma posee un procesador Dual-Core de 32 bits, diversos puertos de comunicación (USB, I2C, SPI, soporte para Ethernet, etc.), cuatro pulsadores y varios LEDs integrados (para poder realizar ciertas pruebas y/o implementaciones con mayor sencillez), ADC y DAC, y varios posibles módulos.

En la Figura 2 pueden observarse todos sus pines con sus nombres y demás conexiones, como por ejemplo USB, junto con los LEDs (LED RGB, LEDs 1 a 3), los botones (TECs 1 a 4), etc.

[2] Recuperado el 19/9/2022 de: https://www.ocompra.com/argentina/item/edu-ciaa-nxp-lpc433-computadora-industrial-abierta-argentina-768178417/

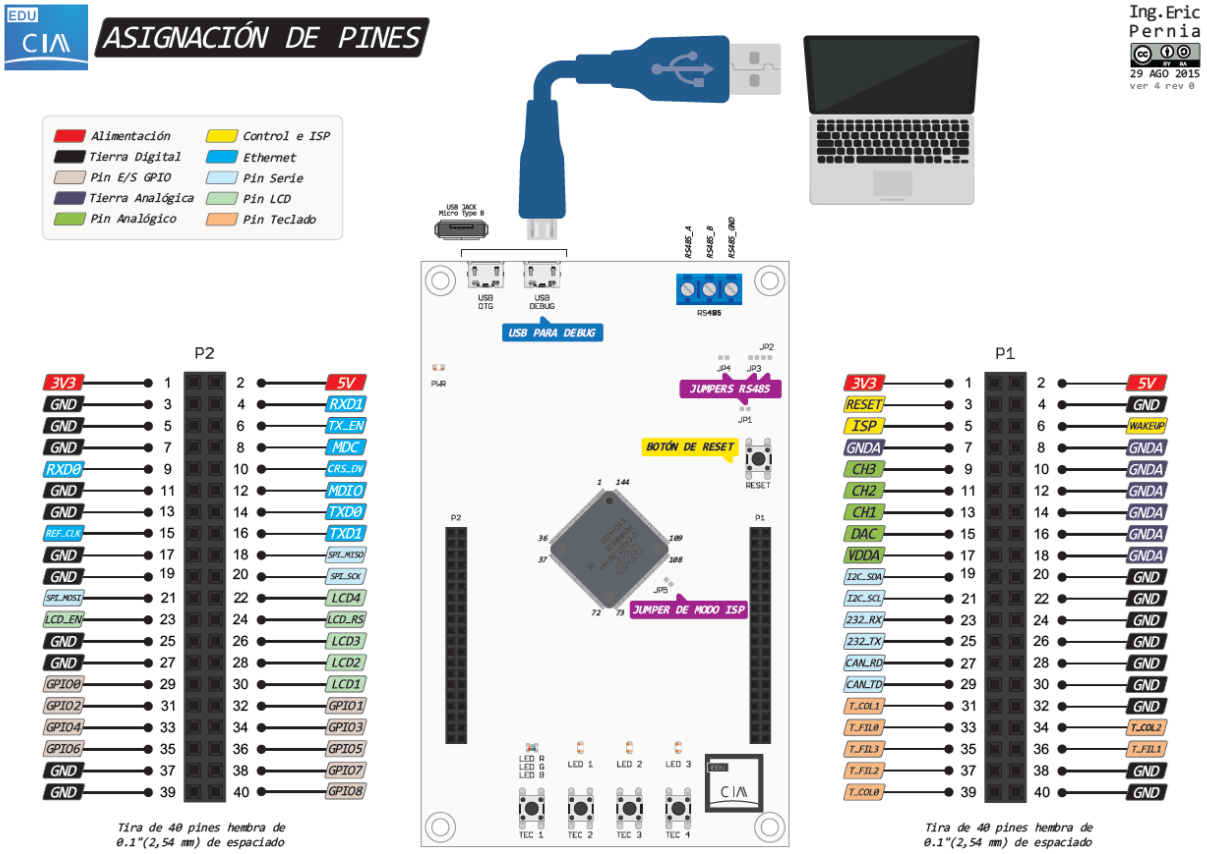

Figura 2[3]

En nuestro caso decidimos programar nuestros proyectos en **C**, a través de **Embedded IDE**, una de las opciones más simples, aunque también puede hacerse mediante otras opciones como Eclipse o Visual.

También utilizamos una librería llamada **sAPI**, la cual nos brindó muchas herramientas indispensables para el manejo del servo motor, el sensor ultrasónico, la configuración de la plaqueta, entre otros. En **gitHub** encontramos varios ejemplos de aplicaciones de esta librería, especialmente para los casos mencionados del servo y el sensor.

Puente H Doble

El driver puente H L298N es el módulo más utilizado para manejar motores DC de hasta 2 amperios. El chip L298N internamente posee dos puentes H completos que permiten controlar 2 motores DC o un motor paso a paso bipolar/unipolar.

El módulo permite controlar el sentido y velocidad de giro de motores mediante señales TTL que se pueden obtener de microcontroladores y tarjetas de desarrollo. El control del sentido de giro se realiza mediante dos pines para cada motor, la velocidad de giro se puede regular haciendo uso de modulación por ancho de pulso (PWM).

Tiene integrado un regulador de voltaje LM7805 de 5V encargado de alimentar la parte lógica del L298N, el uso de este regulador se hace a través de un Jumper y se puede usar para alimentar la etapa de control. En la Figura 3 se adjunta el esquema electrónico del mismo y una imagen para facilitar el desarrollo.

[3] Recuperado el 19/9/2022 de: https://github.com/ciaa/firmware_v3/blob/master/documentation/CIAA_Boards/NXP_LPC4337/EDU-CIAA-NXP/EDU-CIAA-NXP%20v1.1%20Board%20-%202019-01-03%20v5r0.pd

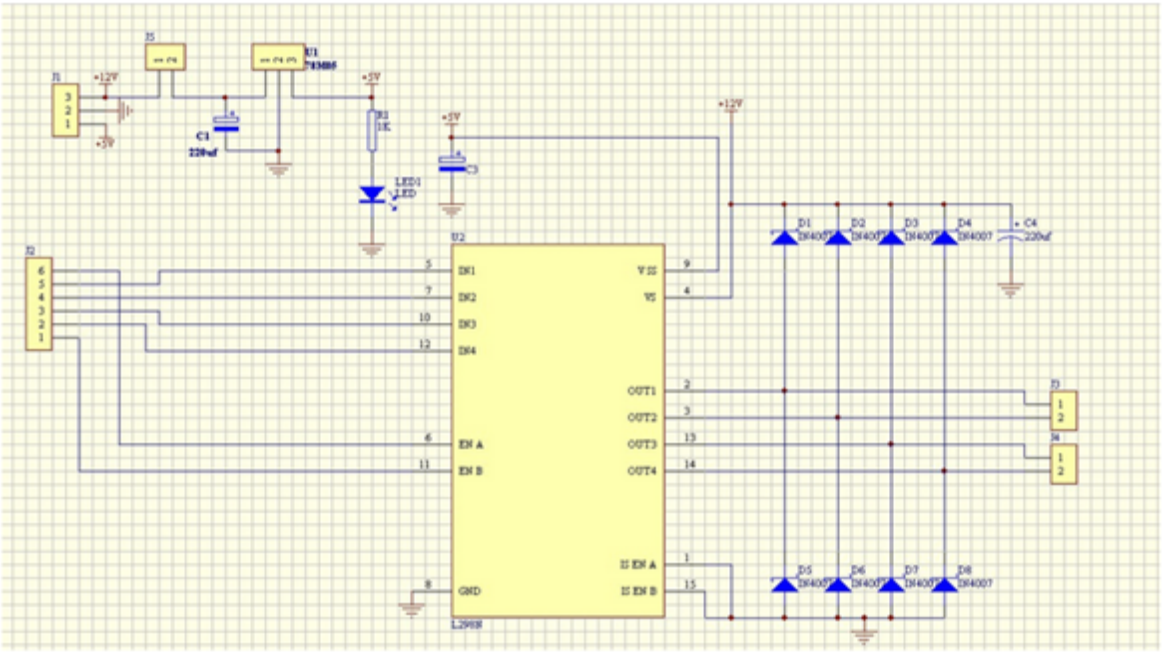

Figura 3

Como puede observarse en la Figura 4, posee un conector de 6 pines para ingresar las señales TTL para controlar los motores, una bornera de tres pines para la alimentación, y dos borneras de 2 pines para la salida a los motores, junto con sus diodos asociados.

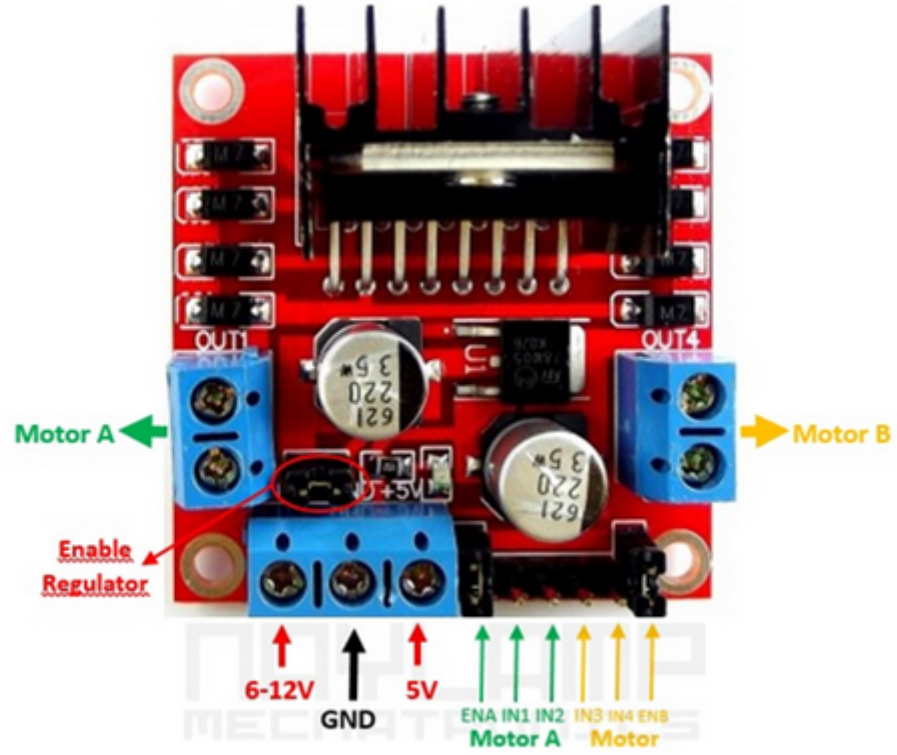

Figura 4

Para poder alimentar el módulo, existen dos formas de hacerlo:

1. Utilizando una sola fuente, conectada a la entrada de 12V y con el Jumper para habilitar el regulador, aclarando que el voltaje de la fuente es el que soporta el motor.

De esta forma la entrada de 5V no debe estar conectada a ninguna fuente, ya que en este pin están presentes 5V a través del regulador interno; pero puedes utilizar este pin como una salida de 5V, pero sin exceder los 500mA de consumo. Se recomienda hacer esta conexión para voltajes menores de 12V para no sobrecalentar el regulador.

2. Utilizando dos fuentes, una de 5V conectada a la entrada de 5V (puede ser los 5V de la placa) y otra fuente con el valor del voltaje que trabaja el motor, conectada al pin de 12V. Para esto se tiene que desconectar el Jumper lo que deshabilitará al regulador.

Para el control del módulo:

Los pines ENA, IN1, IN2 corresponden a las entradas para controlar el MOTOR A (OUT1 y OUT2)

De igual manera ENB, IN3, IN4 permiten controlar el MOTOR B (OUT3 y OUT4)

ENA y ENB, sirven para habilitar o deshabilitar sus respectivos motores, generalmente se utilizan para controlar la velocidad, ingresando una señal de PWM por estos pines. Si no se usan se deben conectar los Jumper para que siempre estén habilitados.

En la Tabla 1, por ejemplo, se puede visualizar los distintos niveles lógicos que se le debe asignar a los pines de control del motor para que este tenga los distintos funcionamientos: avanzar (FORWARD), retroceder (BACKWARD), o detenerse (RELEASE).

AI1	AI2	PWMA	Effect
H	L	H	Motor turns one direction
L	H	H	Motor turns the other direction
L	L	–	Motor stop
H	H	–	Motor stop
–	–	L	Motor stop

Tabla 1[4]

Motor DC

Un motor DC, también conocido como motor de corriente continua, es una máquina que convierte energía eléctrica en energía mecánica, provocando un movimiento rotatorio gracias a la acción de un campo magnético. El circuito equivalente de un motor DC se puede ver en la Figura 5 (es importante mencionar que este circuito es válido para poder mostrar cómo funciona el motor, pero que en la práctica se debe agregar un diodo en paralelo al motor para evitar que la f.e.m. inducida pueda dañar el transistor que lo controla; incluso se podría usar un optoacoplador para evitar que se dañe el microcontrolador); su

[4] Recuperado el 19/9/2022 de: https://itp.nyu.edu/physcomp/labs/motors-and-transistors/dc-motor-control-using-an-h-bridge/

principio de funcionamiento es una espiral de alambre que gira de manera libre en medio del campo de un imán permanente. Cuando por el devanado pasa una corriente, las fuerzas resultantes ejercidas en sus lados y en ángulo recto al campo provocan fuerzas que actúan a cada lado produciendo una rotación. Sin embargo, para que este continúe, cuando el devanado pasa por la posición vertical se debe invertir la dirección de la corriente.

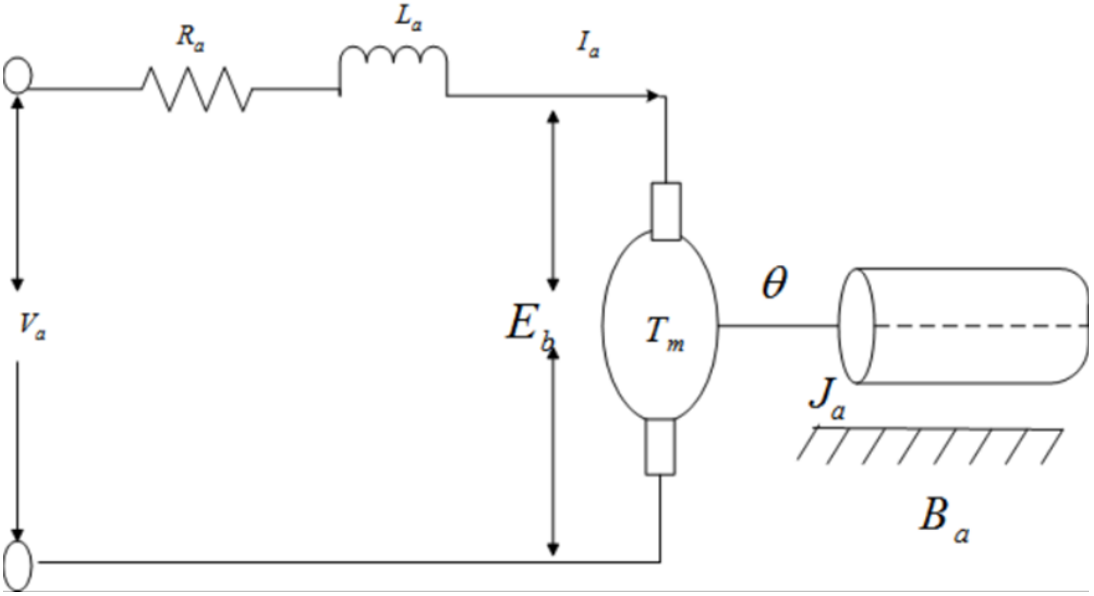

Figura 5

En un motor DC convencional, como se puede ver en la Figura 6, los devanados de alambre se montan en las ranuras de cilindro de material magnético conocido como armadura, la cual está montada en cojinetes (en el campo magnético producido por los polos de campo, que pueden ser imanes permanentes o electroimanes) y puede girar. Conforme la armadura gira, se debe invertir la corriente de cada uno de los devanados al desplazarse por los polos de campo, para así poder asegurar que las fuerzas que actúan en el devanado sigan actuando en la misma dirección y la rotación continúe. La dirección del motor DC se invierte al invertir la corriente de armadura o la corriente de campo.

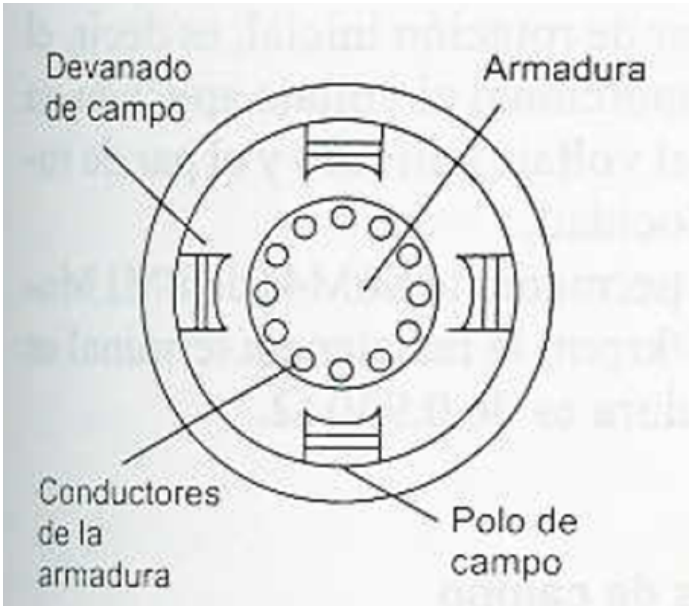

Figura 6

Servomotor

Un servomotor, también conocido más simplemente como servo, es un dispositivo similar a un motor DC que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición. Esto es así porque el servomotor es un motor eléctrico que lleva incorporado un sistema de regulación que puede ser controlado tanto en velocidad como en posición. En la Figura 7 se puede ver el circuito de un servomotor, el cual permite tener un control preciso de la posición lineal o angular, de su velocidad y de su aceleración. Sin embargo, a diferencia del motor de DC, los servomotores sólo pueden girar 180°. Así como los puentes H, los servomotores realizan este control mediante un PWM, donde un pulso mínimo alinea la posición del servo a la izquierda, es decir, a 0°; un pulso máximo alinea la posición del servo hacia la derecha, es decir, a 180°; y los distintos valores entre el pulso mínimo y el máximo determinarán la posición angular que tendrá el servo entre 0° y 180°.

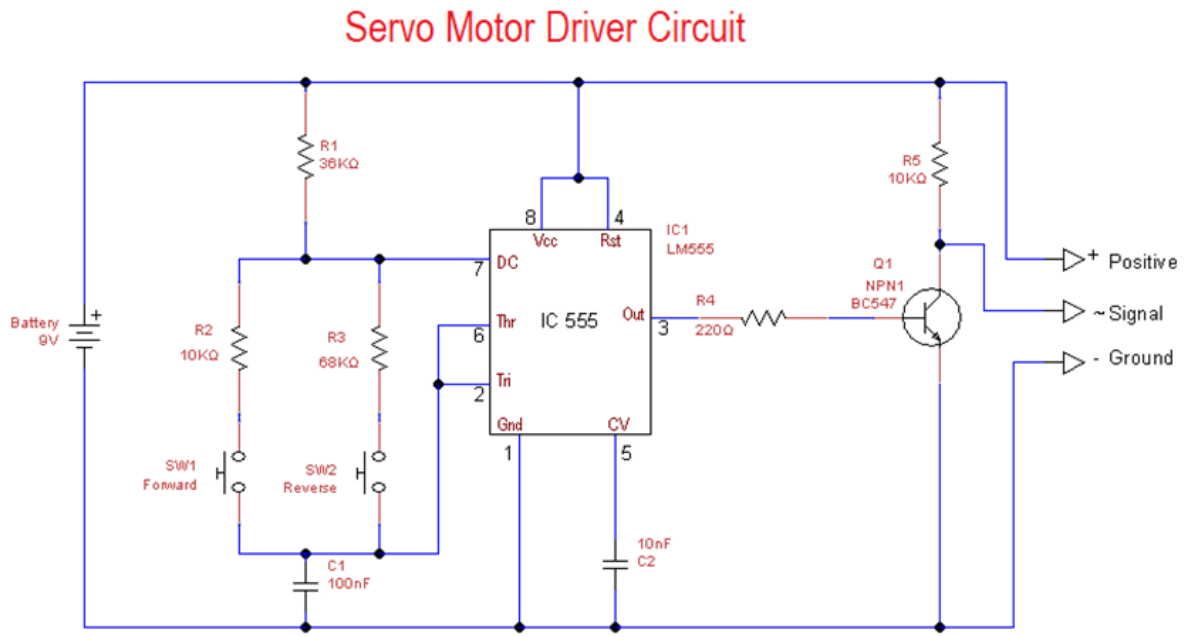

Figura 7

Sensor Ultrasónico

El sensor ultrasónico HC-SR04, mientras tanto, es un sensor de distancia de bajo costo que utiliza ultrasonido para determinar la distancia de un objeto en un rango de 2 a 450 cm. El sensor posee dos transductores: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente: el emisor piezoeléctrico emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIG; después las ondas de sonido viajan en el aire y rebotan al encontrar un objeto, y el sonido de rebote es detectado por el receptor piezoeléctrico; luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto. El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro (aunque los materiales blandos acústicamente como tela o lana pueden llegar a ser difíciles de detectar).

$$\boxed{d = \frac{t * v_s}{2}}$$

Algoritmo 1

La distancia medida por el sensor se puede calcular utilizando la fórmula del Algoritmo 1, donde d es la distancia en metros, t es el tiempo del pulso ECO y v_s es la velocidad del sonido, equivalente a $340\ m/s$. Vale la pena mencionar que nosotros usamos una librería de SAPI que nos resuelve el problema de tener que realizar este cálculo, pero si quisiéramos determinar la distancia en centímetros a partir de pulsos que duran microsegundos, deberíamos tomar la fórmula del Algoritmo 2:

$$\frac{2d}{t} = v_s = \frac{340m}{s} * \frac{1s}{1000000\mu s} * \frac{100cm}{1m} \cong \frac{2cm}{59\mu s} \rightarrow \boxed{d(cm) = \frac{t(\mu s)}{59}}$$

Algoritmo 2

Desarrollo

Para comenzar con el desarrollo de este trabajo de laboratorio, lo primero que hicimos fue investigar las librerías de servomotor y de sensor ultrasónico pertenecientes a sAPI, así como ejemplos de su uso. A partir de eso, comenzamos a escribir en nuestro programa principal un código que permitiera sensar la distancia mediante el sensor ultrasónico y que permitiera a nuestro servo rotar al sensor hacia la izquierda y hacia la derecha para que pueda sensar en esas direcciones. Además, realizamos las conexiones correspondientes de nuestro servomotor y de nuestro sensor ultrasónico. En la Figura 8 se puede ver el diagrama de circuito inicial que nosotros teníamos planeado para poder conectar todos nuestros componentes del sistema; un esquema muy ambicioso que probó ser difícil de implementar, ya que este esquema no nos permitía tener la suficiente cantidad de pines para tener en cuenta la modulación por PWM de la velocidad de los motores.

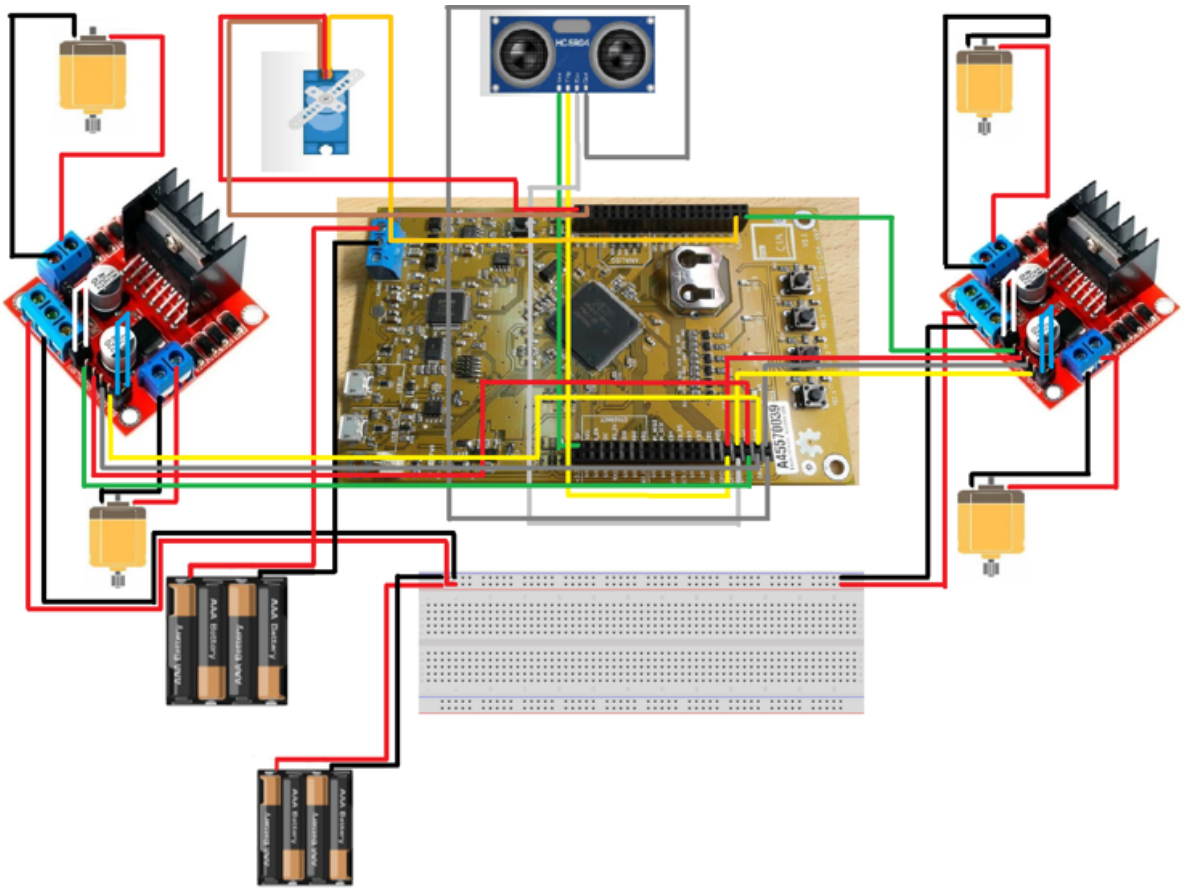

Figura 8

Luego, comenzamos a escribir la lógica para controlar nuestros motores DC, para que así estos puedan ser controlados de una forma eficiente mediante el puente H. Esto lo hicimos mediante la modularización de nuestro código, diseñando un struct motor que tenía definido las conexiones de cada motor y las formas de funcionamiento que podían llevar a cabo: RELEASE, para frenar; FORWARD, para avanzar; y BACKWARD, para retroceder. Un snippet de esto puede ser visto en el Algoritmo 3. Finalmente, y para concluir nuestra primera versión del código, diseñamos un bucle de funcionamiento mediante el cual se permitía que el auto funcione de manera inteligente y autónoma. Cabe destacar que esta primera versión del código puede ser leída en el Apéndice 1.

```
typedef struct {
    uint8_t motorIN1; //Motor pin IN1
    uint8_t motorIN2; //Motor pin IN2
} motor_t;

void motorConfig(motor_t motor, uint8_t _motorIN1, uint8_t _motorIN2){
    gpioConfig(_motorIN1,GPIO_OUTPUT); //Motorn, pin IN1
    gpioConfig(_motorIN2,GPIO_OUTPUT); //Motorn, pin IN2

    motor.motorIN1 = _motorIN1;
    motor.motorIN2 = _motorIN2;
}

void run(motor_t motor, runFunction_t runFunction){

    switch(runFunction){
        case RELEASE:
            gpioWrite(motor.motorIN1, false);
            gpioWrite(motor.motorIN2, false);
            break;

        case FORWARD:
            //gpioWrite(motor.motorIN1, true);
            gpioWrite(T_COL0, true);
            gpioWrite(motor.motorIN2, false);
            printf("IN1: %d .\r\n", gpioRead(T_COL0));
            break;

        case BACKWARD:
            gpioWrite(motor.motorIN1, false);
            gpioWrite(motor.motorIN2, true);
            break;
    }
}
```

Algoritmo 3

Sin embargo, al intentar probar esta lógica, conectando uno de los motores al puente H y este a la placa, nos topamos con nuestro primer gran obstáculo: el sistema no se comportaba como planeábamos. Simplemente, al intentar mandar un true a los pines IN1 o IN2 del motor, los mismos no cambiaban de estado. Continuando con nuestras pruebas, determinamos que el sensor estaba funcionando como correspondía, ya que las lecturas que hacía y que nosotros imprimimos por terminal Serie tenían sentido; y finalmente llegamos a la conclusión de que el problema tenía que estar en el struct. Nuestra teoría principal es que, para que el código del struct funcionase, se necesita usar un tipo de dato especial de mapeo hacia los pines de la EDU-CIAA, el cual nosotros simplemente desconocemos ya que todavía no estamos tan familiarizados con esta tecnología. Otro aspecto importante a considerar es que, como no contábamos con la suficiente cantidad de pines para poder modular la velocidad de los motores, tuvimos que dejar de lado esta funcionalidad, que es muy importante.

Este primer obstáculo entonces nos forzó a cambiar la lógica de nuestro programa, eliminando la modularización que tanto nos habíamos esforzado por incluir y simplificando las funcionalidades del motor para que estén disponibles en el programa principal. Así llegamos a nuestra segunda versión del código, la cual puede ser leída en el Apéndice 2.

Por último, nos quedaba realizar las conexiones necesarias y probar que la lógica de nuestro código sea correcta. Por una cuestión de tiempo, pudimos solo hacer las conexiones necesarias para el uso de dos motores de DC y el sensor (sin los otros dos motores, ni el servo y un puente-H menos), las cuales se esquematizan en la Figura 9, para lo cual también simplificamos nuevamente el código, sacando la configuración y manejo de dichos motores y también la lógica de “mirar” (sensar) hacia derecha e izquierda y cambiar el rumbo hacia la dirección más conveniente. Dicha versión también puede ser leída en el Apéndice 3.

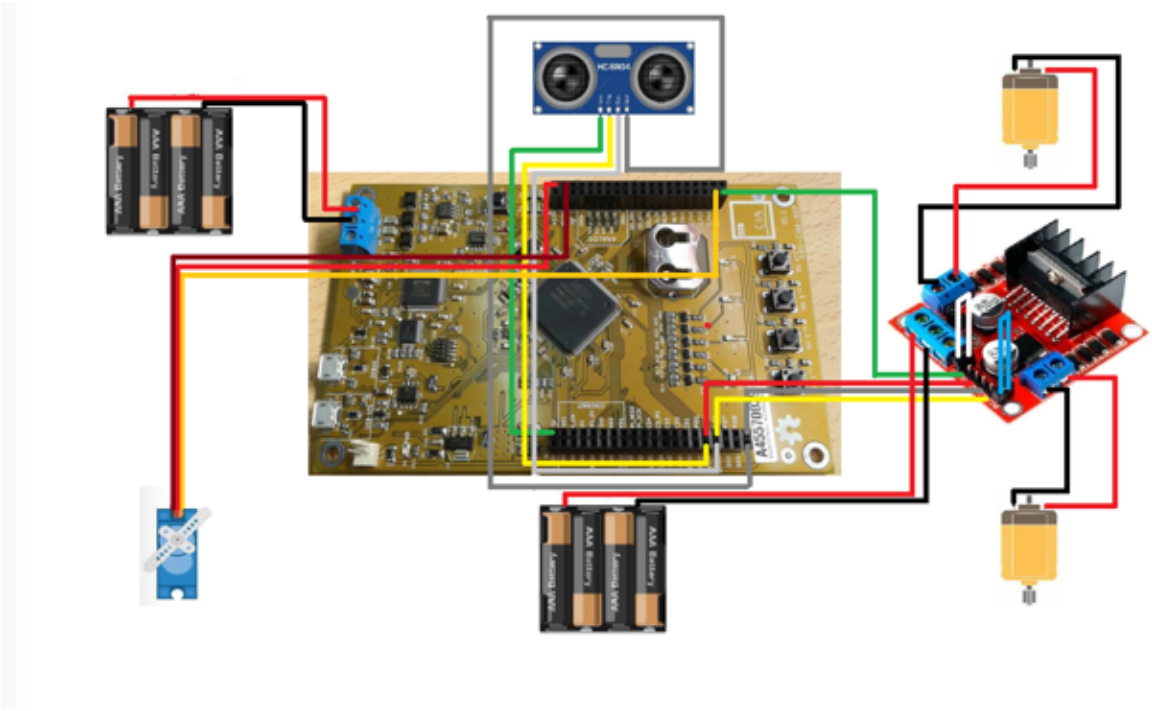

Figura 9

Conclusión

Por desgracia, no pudimos llegar a los resultados deseados en este trabajo de laboratorio, ya que no pudimos hacer andar nuestro concepto inicial de un auto inteligente de cuatro motores DC que emplee un servomotor y un sensor ultrasónico. A la postre, este concepto probó ser demasiado ambicioso y avanzado para nosotros. Quizás con más tiempo y organización lo hubiéramos podido lograr.

Sin embargo, sí pudimos obtener un montón de valiosas lecciones a partir de nuestras pruebas y errores; y, lo más importante, nos pudimos familiarizar un poco con la EDU-CIAA y con las librerías que tiene disponibles.

Esperamos que con el pasar del tiempo, y con muchas más prácticas, podamos desarrollar un conocimiento más extenso de esta tecnología, en la que somos primerizos; y que esto nos permita a su vez tener una mayor facilidad con futuros proyectos y trabajos de laboratorio en los que nos vayamos a embarcar durante el transcurso de esta asignatura.

También, tras esta experiencia, nos proponemos ser más conscientes y eficaces con el uso del tiempo para dividir nuestra idea en porciones más pequeñas y simples para avanzar de una manera más eficiente y sencilla.

Mejoras a Desarrollar

Desafortunadamente, no pudimos hacer andar el concepto inicial de un auto inteligente de cuatro motores DC que use un servomotor y un sensor ultrasónico. Una obvia mejora a desarrollar sería poder tomar este concepto y en efecto crear un prototipo funcional; pero adicionalmente queremos notar que, si hubiéramos tenido más tiempo y recursos, podríamos haber implementado las siguientes mejoras:

- Mejorar la autonomía del auto, la cual hubiera sido muy demandante, ya que necesitaríamos que las baterías den una importante alimentación a los cuatro motores, el servo, el sensor y la placa.
- Agregar un sistema de recarga para las baterías, mediante un cable USB.
- Reemplazar el sistema de cuatro ruedas por una tractor oruga para que el auto pueda desplazarse más fácilmente sobre terrenos irregulares (como un tanque).

Además, el diseño de este auto podría ser cambiado para poder permitir un montón de funcionalidades nuevas, por ejemplo:

- Incorporar una cámara fotográfica para poder grabar el recorrido del auto, haciendo que, en efecto, este sea un astromóvil explorador (como el mars rover). Los vídeos podrían ser guardados en la nube, por lo que necesitaríamos algún módulo de conexión mediante wifi; o bien, un dispositivo de almacenamiento interno. Para eso podríamos también incorporar una linterna y un sensor de luz que, al detectar oscuridad, active la linterna.
- Incorporar un sistema de aspiración y un sensor de suciedad/polvo para hacer que nuestro auto sea esencialmente una aspiradora inteligente. Obviamente también deberíamos agregarle una bolsa para guardar lo aspirado y algún tipo de sistema de sensor + alarma que avise cuando la bolsa esté llena y tenga que ser cambiada.
- Agregar un sensor de sonido y reconocimiento de voz para poder darle instrucciones por voz al auto (como detenerse, explorar, etc.).

Apéndice 1

Adjunta está la versión inicial del código que iba a ser empleado por nuestro sistema, con modularización y el struct de motor.

app.c

```
#include "app.h"          // <= Su propia cabecera
#include "sapi.h"         // <= Biblioteca sAPI (incluye servo y sensor)
#include "motor.h"         // <= Funciones del motor DC

#define SERVO_N SERVO0
motor_t motor1, motor2, motor3, motor4;
bool_t goesForward = false;

int main( void )
{
    //Configuracion PLACA
    boardConfig();
    uartConfig(UART_USB,115200);

    //Configuracion SENSOR
    ultrasonicSensorConfig(ULTRASONIC_SENSOR_0, ULTRASONIC_SENSOR_ENABLE);
    delay(100);

    //Configuracion SERVO
    bool_t value = 0;
    uint8_t servoAngle = 115;
    value = servoConfig(0, SERVO_ENABLE);
    value = servoConfig(SERVO_N, SERVO_ENABLE_OUTPUT);
    value = servoWrite(SERVO_N, servoAngle);
    servoAngle = servoRead(SERVO_N);
    delay(100);

    //Configuracion MOTORES
    motorConfig(motor1, T_COL0, GPIO1); //front left
    motorConfig(motor2, GPIO8, GPIO3);
    motorConfig(motor3, GPIO4, GPIO5); //back left
    motorConfig(motor4, GPIO6, GPIO7);
    //GPIO0 y GPIO2 estan en uso por el sensor, por lo cual usamos en su
lugar T_COL0 y GPIO8 respectivamente

    //Lecturas Iniciales con el sensor
    uint32_t distance;

    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);
```

```
    distance = readPing();
    delay(100);

    while(TRUE){
        printf("Distance: %d cm.\r\n", distance);
        gpioToggle(LEDR); //para chequear que funciona

        int distanceR = 0;
        int distanceL = 0;
        delay(40);

        if(distance <= 15){
            moveStop();
            delay(100);
            moveBackward();
            delay(300);
            moveStop();
            delay(200);
            distanceR = lookRight();
            delay(200);
            distanceL = lookLeft();
            delay(200);

            if(distanceR <= 15 && distanceL <= 15){
                turnAround();
                moveStop();
            } else {
                if(distanceR >= distanceL){
                    turnRight();
                    moveStop();
                }else{
                    turnLeft();
                    moveStop();
                }
            }
        }else{
            moveForward();
        }

        distance = readPing();

        delay(1000);
    };

    return 0;
}

int lookRight(){
    servoWrite(SERVO_N, 50);
    delay(500);
    int distance = readPing();
```

```
    delay(100);
    servoWrite(SERVO_N, 115);
    return distance;
}

int lookLeft(){
    servoWrite(SERVO_N, 170);
    delay(500);
    int distance = readPing();
    delay(100);
    servoWrite(SERVO_N, 115);
    return distance;
}

int readPing(){
    delay(70);
    int cm = ultrasonicSensorGetDistance(ULTRASONIC_SENSOR_0, CM);
    if(cm == 0){
        cm = 250;
    }
    return cm;
}

void moveStop(){
    run(motor1, RELEASE);
    run(motor2, RELEASE);
    run(motor3, RELEASE);
    run(motor4, RELEASE);
    goesForward = false;
  }

void moveForward(){
    if(!goesForward){
        goesForward=true;
        run(motor1, FORWARD);
        run(motor2, FORWARD);
        run(motor3, FORWARD);
        run(motor4, FORWARD);
    }
}

void moveBackward(){
    goesForward=false;
    run(motor1, BACKWARD);
    run(motor2, BACKWARD);
    run(motor3, BACKWARD);
    run(motor4, BACKWARD);
}

void turnRight(){
    goesForward = false;
```

```
    run(motor1, FORWARD);
    run(motor2, BACKWARD);
    run(motor3, FORWARD);
    run(motor4, BACKWARD);
    delay(500);
    moveStop();
    moveForward();
}

void turnLeft(){
    goesForward = false;
    run(motor1, BACKWARD);
    run(motor2, FORWARD);
    run(motor3, BACKWARD);
    run(motor4, FORWARD);
    delay(500);
    moveStop();
    moveForward();
}

void turnAround(){
    goesForward = false;
    run(motor1, FORWARD);
    run(motor2, BACKWARD);
    run(motor3, FORWARD);
    run(motor4, BACKWARD);
    delay(1000);
    moveStop();
    moveForward();
}
```

motor.h

```
#ifndef _DRIVERMOTOR_H_
#define _DRIVERMOTOR_H_

#include "sapi_datatypes.h"

#ifdef __cplusplus
extern "C" {
#endif

typedef struct {
    uint8_t motorIN1; //Motor pin IN1
    uint8_t motorIN2; //Motor pin IN2
} motor_t;

typedef enum{
   RELEASE, FORWARD, BACKWARD
} runFunction_t;
```

```
void motorConfig(motor_t motor, uint8_t _motorIN1, uint8_t _motorIN2);
void run(motor_t motor, runFunction_t runFuction);

#endif
```

motor.c

```
#include "sapi_gpio.h"
#include "motor.h"
#include "sapi.h"

void motorConfig(motor_t motor, uint8_t _motorIN1, uint8_t _motorIN2){
    gpioConfig(_motorIN1,GPIO_OUTPUT); //Motorn, pin IN1
    gpioConfig(_motorIN2,GPIO_OUTPUT); //Motorn, pin IN2

    motor.motorIN1 = _motorIN1;
    motor.motorIN2 = _motorIN2;
}

void run(motor_t motor, runFunction_t runFunction){

    switch(runFunction){
        case RELEASE:
            gpioWrite(motor.motorIN1, false);
            gpioWrite(motor.motorIN2, false);
            break;

        case FORWARD:
            //gpioWrite(motor.motorIN1, true);
            gpioWrite(T_COL0, true);
            gpioWrite(motor.motorIN2, false);
            printf("IN1: %d .\r\n", gpioRead(T_COL0));
            break;

        case BACKWARD:
            gpioWrite(motor.motorIN1, false);
            gpioWrite(motor.motorIN2, true);
            break;
    }
}
```

Apéndice 2

Adjunta está la segunda versión del código que iba a ser empleado por nuestro sistema, sin modularización y simplificando las funciones del struct de motor dentro del programa principal.

app.c

```
#include "app.h"          // <= Su propia cabecera
#include "sapi.h"         // <= Biblioteca sAPI

#define SERVO_N SERVO0
bool_t goesForward = false; //Variable global de control para saber si el
auto está yendo para adelante

int main( void )
{
    //CONFIG PLACA
    boardConfig();
    uartConfig(UART_USB,115200);

    //CONFIG SENSOR
    ultrasonicSensorConfig(ULTRASONIC_SENSOR_0, ULTRASONIC_SENSOR_ENABLE);
    delay(100);

    //CONFIG SERVO
    bool_t value = 0;
    uint8_t servoAngle = 115;
    value = servoConfig(0, SERVO_ENABLE);
    value = servoConfig(SERVO_N, SERVO_ENABLE_OUTPUT);
    value = servoWrite(SERVO_N, servoAngle);
    servoAngle = servoRead(SERVO_N);
    delay(100);

    //CONFIG MOTORES
    gpioConfig(T_COL0,GPIO_OUTPUT); //Motor1 Front Left
    gpioConfig(GPIO1,GPIO_OUTPUT);
    gpioConfig(GPIO8,GPIO_OUTPUT); //Motor2 Front Right
    gpioConfig(GPIO3,GPIO_OUTPUT);
    gpioConfig(GPIO4,GPIO_OUTPUT); //Motor3 Back Left
    gpioConfig(GPIO5,GPIO_OUTPUT);
    gpioConfig(GPIO6,GPIO_OUTPUT); //Motor 4 Back Right
    gpioConfig(GPIO7,GPIO_OUTPUT);
    //GPIO0 y GPIO2 estan en uso por el sensor, por lo cual usamos en su
lugar T_COL0 y GPIO8 respectivamente

    // Hacemos varias lecturas iniciales con el sensor para tener un valor
exacto de distancia inicial
    uint32_t distance;

    distance = readPing();
```

```
    delay(100);
    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);

    while(TRUE){
        printf("Distance: %d cm.\r\n", distance);
        gpioToggle(LEDR); //Para chequear que funciona

        // Declaramos dos variables de distancia, hacia la derecha e
izquierda para cuando el auto deba doblar
        int distanceR = 0;
        int distanceL = 0;
        delay(40);

        if(distance <= 15){
            moveStop();
            delay(100);
            moveBackward();
            delay(300);
            moveStop();
            delay(200);
            distanceR = lookRight();
            delay(200);
            distanceL = lookLeft();
            delay(200);

            if(distanceR <= 15 && distanceL <= 15){
                turnAround();
                moveStop();
            } else {
                if(distanceR >= distanceL){
                    turnRight();
                    moveStop();
                }else{
                    turnLeft();
                    moveStop();
                }
            }
        }else{
            moveForward();
        }

        distance = readPing();
    };

    return 0;
}
```

```
int lookRight(){
    servoWrite(SERVO_N, 50);
    delay(500);
    int distance = readPing();
    delay(100);
    servoWrite(SERVO_N, 115);
    return distance;
}

int lookLeft(){
    servoWrite(SERVO_N, 170);
    delay(500);
    int distance = readPing();
    delay(100);
    servoWrite(SERVO_N, 115);
    return distance;
}

int readPing(){
    delay(70);
    int cm = ultrasonicSensorGetDistance(ULTRASONIC_SENSOR_0, CM);
    if(cm == 0){
        cm = 250;
    }
    return cm;
}

void moveStop(){
    gpioWrite(T_COL0, false);
    gpioWrite(GPIO1, false);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, false);
    gpioWrite(GPIO4, false);
    gpioWrite(GPIO5, false);
    gpioWrite(GPIO6, false);
    gpioWrite(GPIO7, false);
    goesForward = false;
  }

void moveForward(){
    if(!goesForward){
        goesForward=true;
        gpioWrite(T_COL0, true);
        gpioWrite(GPIO1, false);
        gpioWrite(GPIO8, true);
        gpioWrite(GPIO3, false);
        gpioWrite(GPIO4, true);
        gpioWrite(GPIO5, false);
        gpioWrite(GPIO6, true);
        gpioWrite(GPIO7, false);
```

```
    }
}

void moveBackward(){
    goesForward=false;
    gpioWrite(T_COL0, false);
    gpioWrite(GPIO1, true);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, true);
    gpioWrite(GPIO4, false);
    gpioWrite(GPIO5, true);
    gpioWrite(GPIO6, false);
    gpioWrite(GPIO7, true);
}

void turnRight(){
    goesForward = false;
    gpioWrite(T_COL0, true);
    gpioWrite(GPIO1, false);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, true);
    gpioWrite(GPIO4, true);
    gpioWrite(GPIO5, false);
    gpioWrite(GPIO6, false);
    gpioWrite(GPIO7, true);
    delay(500);
    moveStop();
    moveForward();
}

void turnLeft(){
    goesForward = false;
    gpioWrite(T_COL0, false);
    gpioWrite(GPIO1, true);
    gpioWrite(GPIO8, true);
    gpioWrite(GPIO3, false);
    gpioWrite(GPIO4, false);
    gpioWrite(GPIO5, true);
    gpioWrite(GPIO6, true);
    gpioWrite(GPIO7, false);
    delay(500);
    moveStop();
    moveForward();
}

void turnAround(){
    goesForward = false;
    gpioWrite(T_COL0, true);
    gpioWrite(GPIO1, false);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, true);
```

```
    gpioWrite(GPIO4, true);
    gpioWrite(GPIO5, false);
    gpioWrite(GPIO6, false);
    gpioWrite(GPIO7, true);
    delay(1000);
    moveStop();
    moveForward();
}
```

Apéndice 3

Adjunta está la tercera versión del código, la más simple de todas, sin la implementación del servo y de todos los motores de DC (solo dos de ellos).

app.c

```
#include "app.h"          // <= Su propia cabecera
#include "sapi.h"         // <= Biblioteca sAPI

#define SERVO_N SERVO0
bool_t goesForward = false; //Variable global de control para saber si
el auto esta yendo para adelante

int main( void )
{
    boardConfig();
    uartConfig(UART_USB,115200);

    //CONFIG SENSOR
    ultrasonicSensorConfig(ULTRASONIC_SENSOR_0,
ULTRASONIC_SENSOR_ENABLE);
    delay(100);

    //CONFIG SERVO
    bool_t value = 0;
    uint8_t servoAngle = 115;
    value = servoConfig(0, SERVO_ENABLE);
    value = servoConfig(SERVO_N, SERVO_ENABLE_OUTPUT);
    value = servoWrite(SERVO_N, servoAngle);
    servoAngle = servoRead(SERVO_N);
    delay(100);

    //CONFIG MOTORES
    gpioConfig(T_COL0,GPIO_OUTPUT); //Motor1 Front Left
    gpioConfig(GPIO1,GPIO_OUTPUT);
    gpioConfig(GPIO8,GPIO_OUTPUT); //Motor2 Front Right
    gpioConfig(GPIO3,GPIO_OUTPUT);
    gpioConfig(GPIO4,GPIO_OUTPUT); //Motor3 Back Left
    gpioConfig(GPIO5,GPIO_OUTPUT);
    gpioConfig(GPIO6,GPIO_OUTPUT); //Motor 4 Back Right
    gpioConfig(GPIO7,GPIO_OUTPUT);
    //GPIO0 y GPIO2 estan en uso por el sensor, por lo cual usamos en su
lugar T_COL0 y GPIO8 respectivamente

    // Hacemos varias lecturas iniciales con el sensor para tener un
valor exacto de distancia inicial
    uint32_t distance;
```

```
    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);
    distance = readPing();
    delay(100);

    while(TRUE){
        printf("Distance: %d cm.\r\n", distance);
        gpioToggle(LEDR); //Para chequear que funciona

        if(distance <= 15){
            moveStop();
            delay(100);
            moveBackward();
            turnRight();
        }else{
            moveForward();
        }

        distance = readPing();
    };

    return 0;
}

int readPing(){
    delay(70);
    int cm = ultrasonicSensorGetDistance(ULTRASONIC_SENSOR_0, CM);
    if(cm == 0){
        cm = 250;
    }
    return cm;
}

void moveStop(){
    gpioWrite(T_COL0, false);
    gpioWrite(GPIO1, false);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, false);
    goesForward = false;
  }

void moveForward(){
    if(!goesForward){
        goesForward=true;
        gpioWrite(T_COL0, true);
```

```
        gpioWrite(GPIO1, false);
        gpioWrite(GPIO8, true);
        gpioWrite(GPIO3, false);
    }
}

void moveBackward(){
    goesForward=false;
    gpioWrite(T_COL0, false);
    gpioWrite(GPIO1, true);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, true);
}

void turnRight(){
    goesForward = false;
    gpioWrite(T_COL0, true);
    gpioWrite(GPIO1, false);
    gpioWrite(GPIO8, false);
    gpioWrite(GPIO3, true);
    delay(500);
    moveStop();
    moveForward();
}
```

Bibliografía

Bolton, W. (2013). *Mecatrónica: Sistemas de Control Electrónico en la Ingeniería Mecánica y Eléctrica. Quinta Edición.* Alfaomega Grupo Editor. México.

Computadora Industrial Abierta Argentina. Proyecto CIAA. Recuperado el 19/9/2022 de: http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=start

Computadora Industrial Abierta Argentina. SASE. Recuperado el 19/9/2022 de: http://www.sase.com.ar/asociacion-civil-sistemas-embebidos/ciaa/

Driver L298N Puente H. Rambal. Recuperado el 19/9/2022 de: https://rambal.com/drivers/866-driver-l298n.html

EDU-CIAA-NXP Esquemático [Archivo pdf]. GitHub. Recuperado el 19/9/2022 de: https://github.com/ciaa/firmware_v3/blob/master/documentation/CIAA_Boards/NXP_LPC4337/EDU-CIAA-NXP/EDU-CIAA-NXP%20Esquematico.pdf

EDU-CIAA-NXP Información general [Archivo pdf]. GitHub. Recuperado el 19/9/2022 de: https://github.com/ciaa/firmware_v3/blob/master/documentation/CIAA_Boards/NXP_LPC4337/EDU-CIAA-NXP/EDU-CIAA-NXP%20Informaci%C3%B3n%20general.pdf

EDU-CIAA-NXP Mapeo de pines Firmata4CIAA v2 [Archivo pdf]. GitHub. Recuperado el 19/9/2022 de: https://github.com/ciaa/firmware_v3/blob/master/documentation/CIAA_Boards/NXP_LPC4337/EDU-CIAA-NXP/EDU-CIAA-NXP%20Mapeo%20de%20pines%20Firmata4CIAA%20v2.pdf

EDU-CIAA-NXP v1.1 Board - 2019-01-03 v5r0 [Archivo pdf]. GitHub. Recuperado el 19/9/2022 de: https://github.com/ciaa/firmware_v3/blob/master/documentation/CIAA_Boards/NXP_LPC4337/EDU-CIAA-NXP/EDU-CIAA-NXP%20v1.1%20Board%20-%202019-01-03%20v5r0.pdf

Historia de la ASCE. SASE. Recuperado el 19/9/2022 de: http://www.sase.com.ar/asociacion-civil-sistemas-embebidos/

Motor de corriente continua. Wikipedia. Recuperado el 19/9/2022 de: https://es.wikipedia.org/wiki/Motor_de_corriente_continua

Lab: DC Motor Control Using an H-Bridge. ITP Physical Computing. Recuperado el 19/9/2022 de:*https://itp.nyu.edu/physcomp/labs/motors-and-transistors/dc-motor-control-using-an-h-bridge/*

SENSOR ULTRASONIDO HC-SR04. Naylamp Mechatronics. Recuperado el 19/9/2022 de: https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

Servomotor. Wikipedia. Recuperado el 19/9/2022 de: https://es.wikipedia.org/wiki/Servomotor

Proyecto CIAA. Proyecto CIAA. Recuperado el 19/9/2022 de: http://www.proyecto-ciaa.com.ar/

Repositorio sAPI. GitHub. Recuperado el 19/9/2022 de: https://github.com/epernia/firmware_v3